In Proceedings of the Nonlinear Theory and Applications Symposium (NOLTA), 1993, Hawaii.

DISCRETE RECURRENT NEURAL NETWORKS AS PUSHDOWN AUTOMATA

Zheng Zeng[†], Rodney M. Goodman[†] and Padhraic Smyth[††]

[†]Department of Electrical Engineering, 116-81
California Institute of l'ethnology
Pasadena, CA 91125, U.S.A.

[††]Jet Propulsion Laboratory, 238-420
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA91109, U.S.A.

In this paper we describe a new discrete recurrent neural network model with discrete external stacks for learning context-free grammars (or pushdown automata). Conventional analog recurrent networks tend to have stability problems when presented with input sirings which are longer than those used for training: the network's internal states become merged and the string can not be correctly parsed. However, the discrete recurrent structure forms a *stable* representation during learning by using isolated discrete points as its internal representation of states for the automata. Hence, once successfully trained, the network is perfectly stable on input strings of arbitrary length. For training such discrete networks a novel "pseudo-gradient" learning rule is used, Experimental results demonstrate the ability of the discrete network to learn context-free grammars in a stable manner. The discrete network model results in the advantages of a stable network, a clear understanding of the operation of the stack, and a structure which is easily implementable in hardware.

1. INTRODUCTION

We consider the problem of learning context-free grammars from labeled examples using recurrent networks. Analog recurrent networks have recently been shown to have the ability to learn context-free grammars by using an external "continuous stack" [1], We have shown in our previous research [3] that analog recurrent networks have difficulty in forming stable internal state representations for grammar learning, i.e., after successful training, as the lengths of the test strings get longer and longer, the network tends to "forget" which state it is in and performance deteriorates significantly.

The problem is inherent to the internal representation of any network which uscs analog values to represent states, while the states in the underlying state machine are actually discrete.

To achieve stability for long strings, wc propose a discrete recurrent network structure which uses discretization in both its feedback links and in the operation of an external discrete stack,

2. NETWORK ARCHITECTURE

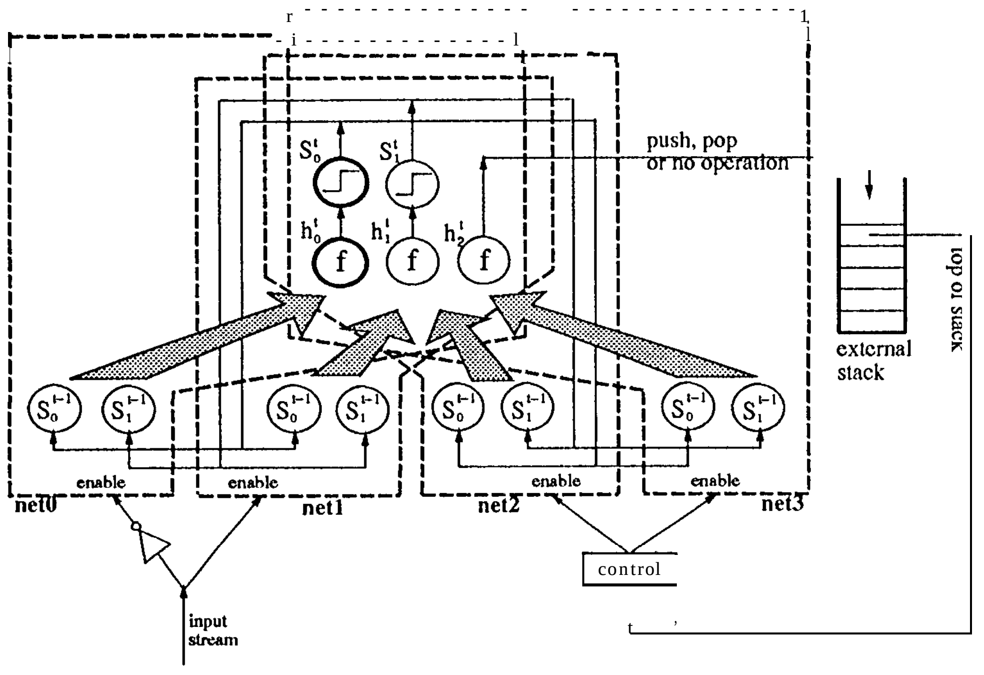


Figure **1.** A discretized second-order network with an external stack. The thick circled unit h_0^t is the indicator unit: h_0^t >0.5 for legal sirings and h_0^t <0.5 for illegal strings.

A second-order discrete recurrent network with an external stack for the case of binary input and stack alphabets is shown in Fig. 1. The primary differences between this structure and the one proposed in [1] are that wc have (a) a discrete stack and (b) discretized units. The network is represented as four separate networks with shared hidden units, controlled by 2 gating switches: the input symbol enables or disables **net0** and netl, and the top-of-stack symbol enables or disables net2 and **net3**. The common hidden unit values are discretized and copied back to all 4 "subnetworks" after each time step. The hidden unit activation function is the standard sigmoid function, $f(x)=\frac{1}{1+e^{-x}}$. The discretization function is defined to be:

$$D(x)=\begin{cases}0.8 & \text{if } x\geq 0.5\\ 0.2 & \text{if } x<0.5\end{cases}. \quad (1)$$

Hidden unit h_0 is chosen to be a special indicator unit whose activation should bc greater than 0.5 at the end of a legal string, or smaller than 0.5 otherwise. The last hidden unit, in this case h_2, is designated to bc the "action" unit, whose activation decides what stack action to take. The network weights are initialized randomly with a uniform distribution between -1 and 1.

As in [1], we restrict the scope of context-free grammars as follows; the alphabet of the stack symbol is set to be the same as the input alphabet, only the current input symbol can be pushed onto the stack, and epsilon transitions (which can make state transitions or stack actions without reading in a new input symbol) are not allowed.

•

3. ERROR FUNCTIONS

Several situations can be encountered during learning, each requiring the use of a different error function. Let $h_0, h_1, \ldots. h_N$ be the hidden units of the network, where ho is the "indicator" unit and h_N *is* the "action" unit. Let L be the length of the current string being processed, and d^t be the depth of the stack at time step t (hence, d^L is the depth of the stack at the end of the string).

Das et al. have suggested in [1] that learning can be sped up significantly by providing the network with a "teacher" to give hints. Whenever a point is reached in the input string such that the string up to that point is not a prefix of any legal strings, the teacher produces a signal and the network is trained to have another special hidden unit, designated as the "dead unit ," turn on. Our error functions are similar in general to those proposed in [1] but there are some significant differences [4]. For example, for the case when the string is illegal but not a dead string, and the end of a string is reached (without any attempt to pop an empty stack), the error function is defined to bc:

$$E = \begin{cases} h_0^L - d^L + \frac{1}{2}(h_1^L)^2 & \text{if } h_0^L - d^L > 0 \\ 0 & \text{otherwise,} \end{cases}$$

i.e., we want either the stack to be nonempty, or the indicator unit to be off, *and for both cases, the dead unit to be off.* The dead unit should not be on for such strings because they could be prefixes to legal strings.

A detailed description of error functions used for various cases can be found in [4]. Experiments were carried out for learning both with and without hints. For the case of learning without hints, the error functions can bc easily modified by canceling any term concerning the "dead" unit, h_l, and cases concerning the teacher signal will not be encountered.

4. THE PSEUDO-GRADIENT METHOD

To train the discretized network, we propose an approximation to gradient descent which we call the pseudo- gradient learning rule [3], 'l'he essence of the learning rule is that in doing gradient descent it makes use of the gradients of a sigmoid function as heuristic hints in place of those of the hard-limiting function, while still using the discretized values in the feedback update paths and in the operations on the external stack,

During training, at the end of each string $\{x^0, x^1, \ldots. x^L\}$ the error is calculated according to the definition for the appropriate case.

Update w_{ij}^n, the weight from unit j to unit i in net$n$, at the end of each string presentation: $w_{ij}^n = w_{ij}^n - \alpha\frac{\partial E}{\partial w_{ij}^n}$, for all n, i, j. For the example case of Section 3, we have:

$$\frac{\widetilde{\partial E}}{\partial w_{ij}^n} = \begin{cases} \frac{a\text{-}h:}{\partial w_{ij}^n} - \frac{\tilde{\partial} d^L}{\partial w_{ij}^n} + h_1^L \frac{\bar{\partial} h_1^L}{\partial w_{ij}^n} & \text{if } h_0^L - d^L > 0 \\ 0 & \text{otherwise,} \end{cases} \qquad \forall n, \text{i, j,}$$

where $\frac{\tilde{\partial}}{\partial w_{ij}^n}$ is what we call the "pseudo-gradient" with respect to w_{ij}^n.

To get the pseudo-gradients $\frac{\tilde{\partial} h_0^L}{\partial w_{ij}^n}$ and $\frac{\tilde{\partial} h_1^L}{\partial w_{ij}^n}$, pseudo-gradients $\frac{\tilde{\partial} h_k^t}{\partial w_{ij}^n}$ for all t, k need to be calculated forward in time at each time step. Initially, set: $\frac{\tilde{\partial} h_k^0}{\partial w_{ij}^n}$ = O, for all i, j, n, k. For weights in the input-controlled subnetworks:

$$\frac{\widetilde{\partial h}_k^t}{\partial w_{ij}^n} = f' \cdot (\sum_l w_{kl}^{x^t} \frac{\widetilde{\partial h}_l^{t-1}}{\partial w_{ij}^n} + \delta_{ki}\delta_{nx^t} S_j^{t-1}), \quad \forall k, t, w_{ij}^n. \tag{2}$$

I.e., in carrying out the chain rule for the gradient we replace the real gradient $\frac{\partial S_l^{t-1}}{\partial w_{ij}^n}$, which is zero almost everywhere, by the pseudo-gradient $\frac{\widetilde{\partial h}_l^{t-1}}{\partial w_{ij}^n}$.

To obtain the term $\frac{\widetilde{\partial d}^t}{\partial w_{ij}^n}$, we use the iterative operational equation:

$$d^t = d^{t-1} + D_1(h_N^t), \qquad where \ D_1(x) = \begin{cases} 1 & \text{if } x > 0.4 \\ -1 & \text{if } x < 0.6 \\ 0 & \text{otherwise.} \end{cases}$$

Initially, set $\frac{\widetilde{\partial d}^0}{\partial w_{ij}^n}$ = O for all n, i, j. After each time step, update:

$$\frac{\widetilde{\partial d}^t}{\partial w_{ij}^n} = \frac{\widetilde{\partial d}^{t-1}}{\partial w_{ij}^n} + \frac{\widetilde{\partial h_N}^t}{\partial w_{ij}^n}, \qquad \forall n, i, j.$$

Here, in place of the gradient of the piece-wise step function D_1, we still use the pseudo-gradient of the action unit h_N. Although the value of the action unit does not get discretized and copied back after each time step, its pseudo-gradient can still be calculated by utilizing the pseudo-gradients of other hidden units. For weights in the input-controlled subnetworks:

$$\frac{\widetilde{\partial h}_N^t}{\partial w_{ij}^n} = f' \cdot (\sum_{l=0}^{N-1} w_{Nl}^{x^t} \frac{\widetilde{\partial h}_l^{t-1}}{\partial w_{ij}^n} + \delta_{Ni}\delta_{nx^t} S_j^{t-1}), \quad \forall i, j, n, t.$$

Similar equations can be derived for weights in the top-of-stack-controlled subnetworks.

In the formulae, we have left out a term concerning the top-of-stack symbol's dependency on the weights. Since a simple recurrent form of this term is analytically impossible to derive, an approximation was used in [1]. In our formula, the pseudo-gradient is itself an approximation, further fine tuning by this term may not be necessary. Empirical results in the next section will demonstrate that the networks can indeed perform successful learning without this term in the formula. Thus, the coupling between the stack and the network during learning is reflected only in the previous formula for the gradient of the stack depth.

5. EXPERIMENTAL RESULTS

We experimented with the same grammars as used in [I], i.e.,

- The parenthesis matching grammar.
- The postfix grammar.
- $a^n b^n$.
- $a^{m+n} b^m c^n$.
- $a^n b^n c b^m a^m$.

Table 1 (a) and (b) show the detailed results for experiments with and without hints, respectively. From the results in Table 1, it can be seen that providing the network with hints can indeed speed up learning, or even enable the learning of the grammars in cases where the grammar could not be learned without hints .

Table 1
Experimental results from training the discrete recurrent net work on context-free grammars (a) with hints; (b) without hints. The training set and hidden unit columns indicate the fixed learning parameters for each grammar. 10 runs with different random initial weights were carried out for each grammar except for the $a^nb^ncb^ma^m$ grammar in (a) and a^nb^n in (b), for which only one run each was obtained. N_s, the number of successful runs is the number of runs (of the 10 possible) for which the trained network generalized perfectly for strings of any length. The means for the epochs and total characters processed (and the standard deviation for the epochs) were estimated only from the successful runs. NO, the number of over-fitting runs is the number where the network over-fitted the data and did not generalize perfectly. N_n, the number of non-convergent runs is the number of runs where the network did not converge on the training data after 1000 epochs.

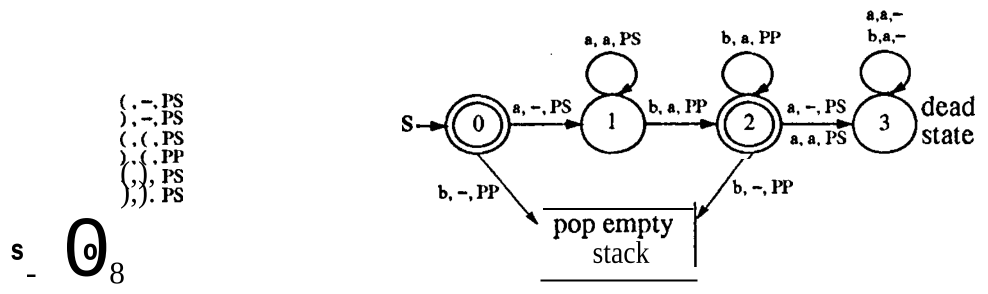
grammar	*training set* # of strings	*training set* L_{max}	# of hidden units	N_n	NO	N_s	mean # of epochs	σ of epochs	mean # of total characters
Parenthesis	46	6	3	0	0	10	28.8	16.3	5205
Postfix	63	7	4	1	0	- 9	62.3	17.1	21131
anbn	32	6	4	2	0	- 8	127.3	4.9	16797
$a^{m+n}b^mc^n$	120	8	5	2	0	8	6 3	36.0	7560
$a^nb^ncb^ma^m$	150	7	7	—.		1	698	--	516520

(a)

grammar	*training set* # of strings	*training set* L_{max}	# of hidden units	N_n	No	N_s	mean # of epochs	O of epochs	mean # of total characters
Parenthesis	180	6	3	0	0	10	12.0	10.5	11208
Postfix	371	7	4	4	2	4	185.8	149.0	408464
an b^n	760	8	5	—	–	- 1	6 3	- - -	332136

(b)

As an example, Fig. 2(a) and (b) show the derived pushdown automata from the networks after being trained on the parenthesis matching grammar and the a^nb^n grammar respectively.
Once the network has successfully learned the pushdown automata from the training set, its internal states are always stable, i.e., it has 100% classification rate on unseen strings of arbitrary length. In contrast, in [1], where a continuous stack was used, the results show that analog recurrent networks do not always generalize perfectly.



(a) (b)

Figure 2. Extracted pushdown automata from the discretized network with an external stack after learning (a) the parenthesis grammar without hints; (b) the grammar $a^n b^n$ with hints. Double circled means the state has an indicator unit on, SO=O.8: thus a processed string is legal if the automaton arrives at such a state *and* if the stack is empty. A dead state means the state has its dead unit on, $S_1 = 0.8$: a processed string is illegal as soon as the automaton arrives at such a state. A transition rule is labeled by "x,y,z", where x stands for the current input symbol, y stands for the top-of-stack symbol ("-" means an empty stack), and z stands for the operation taken on the stack: "PS" means push, "PP" means pop.

6. CONCLUSION

In this paper we introduced a discrete recurrent network structure with an external stack for the task of learning pushdown automata. The discrete structure results in the advantages of a stable network, a clear understanding of the operation of the stack, and can be easily implemented in hardware.

Acknowledgments

The research described in this paper was supported in part by ONR and ARPA under grant number NOO014-92-J-1860. In addition this work was carried out in part by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

1. S. Das, C. I,. Gilts, G. Z. Sun, "Using prior knowledge in an NNPDA to learn context-free languages," *Advances in Neural Information Processing Systems 5, S.* J. Hanson, J. D. Cowan and C.L. Giles, Eds., San Mateo, CA: Morgan Kaufmann, pp.65-72, 1993.
2. J. E. Hopcroft, J. D. Unman, *Introduction to Automata Theory, Languages and Computation,* Addison-Wesley, Reading Mass., 1979.
3. Z. Zeng, R. Goodman, P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation,* Vol. 5, No. 6, pp.976-990, 1993.
4. Z. Zeng, R. Goodman, P. Smyth, "Discrete recurrent neural networks for grammatical in ference," to appear in *IEEE Transactions on Neural Networks,* Vol. 4, No. 6, 1993.